**Cost Optimization Proposal for Azure Cosmos DB (Read-Heavy Workload)**

**Problem Statement**

A significant portion of data stored in Azure Cosmos DB becomes rarely accessed after 3 months. Despite being infrequently used, it continues to incur high storage and RU/s costs, impacting overall cloud expenditure.

**Proposed Solution: Hybrid Data Tiering Strategy**

**Overview:**  
Implement a **time-based data tiering** mechanism:

* Keep **recent records (≤ 3 months)** in Azure Cosmos DB (Hot Tier).
* Move **older records (> 3 months)** to **Azure Blob Storage** (Cold Tier).
* Provide **transparent API-level access** to both hot and cold data, ensuring **no contract changes**.

**Benefits**

| **Benefit** | **Description** |
| --- | --- |
| **Cost Reduction** | Up to **60–80% cost savings** on storage and RU/s for older data. |
| **No Data Loss** | Old data is **archived, not deleted**. |
| **Seamless Access** | API can be abstracted to **route requests** based on timestamp without breaking existing clients. |
| **No Downtime** | Migration and access work **in the background**. |
| **Optimized Performance** | Cosmos DB retains only hot, high-demand data → improved RU utilization. |

**Estimated Cost Savings**

| **Metric** | **Cosmos DB** | **Blob Storage** |
| --- | --- | --- |
| Storage per GB | ₹20–₹25 | ₹1.5–₹2 |
| Read Charges | RU/s charges | On-demand |
| Cost Reduction | ~80% older data archived | Cosmos DB usage cut in half |

**Example:**  
If 1 TB of data is stored in Cosmos DB and 800 GB is older than 3 months:

* **Before:** ₹25,000/month for 1 TB
* **After:** ₹5,000 (Cosmos DB) + ₹1,600 (Blob) = **~₹6,600/month**
* **Savings:** Over 70%

**Implementation Architecture**

|  |
| --- |
| Client/API Consumers |

|  |
| --- |
| API Gateway /Azure Function |

(recent data) (archived data)

|  |
| --- |
| Azure Blob Storage (Cold) |

|  |
| --- |
| Azure CosmosDB (Hot Tier) |

**Key Features to Enable**

1. **Time-based Archival Script or Function** (e.g., every night):

|  |
| --- |
| if record.timestamp < (today - 90 days):  move\_to\_blob(record)  delete\_from\_cosmos(record) |

1. **Query Redirection Logic in API**:
   * If date > (today - 90 days): fetch from Cosmos DB
   * Else: fetch from Blob Storage
2. **Soft Delete / PITR in Cosmos DB** *(initial rollout only)*

**Optional Enhancements (Bonus)**

| **Need** | **Solution** |
| --- | --- |
| **Search in archived data** | Use **Azure Cognitive Search** or **Synapse + Data Lake** |
| **Tier further within Blob** | Use **Blob Lifecycle Management** to move old blobs from Hot → Cool → Archive |
| **Data Retention Governance** | Implement **Data Retention Policy** using tags and deletion triggers |
| **Monitoring & Alerts** | Azure Monitor / Log Analytics integration for API errors or delays |
|  |  |

**No Changes to API Contracts**

* All logic is **abstracted inside API gateway** (e.g., Azure Function, API Management).
* Front-end or client applications remain **unaffected**.

**Conclusion**

Implementing a hybrid data tiering strategy using **Azure Cosmos DB + Blob Storage** is a simple yet highly effective solution that:

* Drastically **reduces cloud costs**
* Preserves **data availability**
* Maintains **API compatibility**
* Requires **minimal architectural changes**

Below is a **collection of pseudocode, commands, and scripts** that implement the **core logic** for the **hybrid cost optimization strategy** with **data archival, retrieval, and automation**, focusing on **Azure Cosmos DB → Azure Blob Storage**, while keeping **API contract unchanged** and ensuring **no downtime or data loss**.

**Note:** I have taken help of Chatgpt AI to create scripts.

1. **Pseudocode – Archival Script (Cosmos DB → Blob Storage)**

|  |
| --- |
| import datetime  from azure.cosmos import CosmosClient  from azure.storage.blob import BlobServiceClient  import json  # Configuration  COSMOS\_URI = "https://<cosmos-account-name>.documents.azure.com:443/"  COSMOS\_KEY = "<cosmos-key>"  DATABASE\_NAME = "billingdb"  CONTAINER\_NAME = "records"  BLOB\_CONN\_STR = "<blob-connection-string>"  BLOB\_CONTAINER = "archive-records"  # Initialize clients  cosmos\_client = CosmosClient(COSMOS\_URI, COSMOS\_KEY)  container = cosmos\_client.get\_database\_client(DATABASE\_NAME).get\_container\_client(CONTAINER\_NAME)  blob\_service = BlobServiceClient.from\_connection\_string(BLOB\_CONN\_STR)  blob\_container = blob\_service.get\_container\_client(BLOB\_CONTAINER)  # Archive threshold  threshold\_date = datetime.datetime.utcnow() - datetime.timedelta(days=90)  # Query Cosmos DB for old records  query = f"SELECT \* FROM c WHERE c.timestamp < '{threshold\_date.isoformat()}'"  for record in container.query\_items(query=query, enable\_cross\_partition\_query=True):  record\_id = record['id']    # Upload to Blob Storage  blob\_name = f"{record\_id}.json"  blob\_container.upload\_blob(blob\_name, json.dumps(record), overwrite=True)  # Optionally delete from Cosmos DB to reduce cost  container.delete\_item(item=record\_id, partition\_key=record['partitionKey']) |

1. **Retrieval Logic (Abstracted in API)**

|  |
| --- |
| def get\_record(record\_id, timestamp):  if is\_recent(timestamp):  # Fetch from Cosmos DB  return get\_from\_cosmos(record\_id)  else:  # Fetch from Blob  return get\_from\_blob(record\_id)  def is\_recent(timestamp):  archive\_threshold = datetime.datetime.utcnow() - datetime.timedelta(days=90)  return timestamp > archive\_threshold  def get\_from\_cosmos(record\_id):  # Logic to get document from Cosmos DB  ...  def get\_from\_blob(record\_id):  blob\_client = blob\_service.get\_blob\_client(container=BLOB\_CONTAINER, blob=f"{record\_id}.json")  return json.loads(blob\_client.download\_blob().readall()) |

1. **Schedule Archival with Azure Function (Timer Trigger)**

|  |
| --- |
| **{**  "bindings": [  {  "name": "timer",  "type": "timerTrigger",  "direction": "in",  "schedule": "0 0 \* \* \* \*" // every day at midnight  }  ]  } |

Python code (run\_daily\_archiver/**init**.py)

|  |
| --- |
| def main(timer: func.TimerRequest) -> None:  logging.info("Running daily archival...")  archival\_logic() # same as the logic above |

1. **Cosmos DB Configuration for Autoscale**

|  |
| --- |
| # Create Cosmos DB container with autoscale (CLI)  az cosmosdb sql container create \  --account-name myaccount \  --database-name billingdb \  --name records \  --partition-key-path "/partitionKey" \  --max-throughput 4000 |

1. **Blob Storage Lifecycle Policy (Move to Archive Tier)**

Sample lifecycle.json:

|  |
| --- |
| **{**  "rules": [  {  "name": "move-to-archive",  "enabled": true,  "type": "Lifecycle",  "definition": {  "filters": {  "blobTypes": ["blockBlob"],  "prefixMatch": ["archive-records/"]  },  "actions": {  "baseBlob": {  "tierToCool": { "daysAfterModificationGreaterThan": 30 },  "tierToArchive": { "daysAfterModificationGreaterThan": 90 }  }  }  }  }  ]  } |

Apply with CLI:

|  |
| --- |
| az storage account management-policy create \  --account-name mystorageacct \  --policy @lifecycle.json \  --resource-group myrg |

1. **Cosmos DB TTL (Time-to-Live) Setup**

|  |
| --- |
| az cosmosdb sql container update \  --account-name myaccount \  --database-name billingdb \  --name records \  --ttl -1 # Enables TTL (must set \_ts or TTL field in document) |

In documents:

|  |
| --- |
| {  "id": "123",  "value": "...",  "ttl": 7776000 # 90 days in seconds  } |

## ****7. Query Index Optimization (Disable Indexing for Archived Data)****

**Create container with custom indexing policy (disable for archived records)**

|  |
| --- |
| "indexingPolicy": {  "automatic": true,  "indexingMode": "consistent",  "includedPaths": [  {  "path": "/\*"  }  ],  "excludedPaths": [  {  "path": "/archived/\*"  }  ]  } |